Università degli Studi di Parma
Dipartimento di Ingegneria e Architettura
Sistemi operativi e in tempo reale – a.a. 2022/23

Scheduling di job aperiodici e sporadici

In scheduling priority-driven di task periodici

Introduzione

- Scenario realistico: compresenza di task “misti”
- Task:
  - Periodici -> (generalmente) hard RT
  - *Aperiodici* (in senso lato):
    - Hard RT (task sporadici)
    - Soft RT
    - Non RT
- I task periodici sono in genere gestiti con uno degli algoritmi già presentati, indipendentemente dalla presenza di task aperiodici
- Tipicamente: RM o DM se a priorità statica, EDF se a priorità dinamica

Introduzione

- Ipotesi:
  - Task periodici rilasciati simultaneamente all'istante t=0
  - Task e job aperiodici rilasciati in istanti arbitrari, non noti a priori

- Obiettivo:
- Garantire i task periodici senza penalizzare eccessivamente i task aperiodici:
  - Per i task aperiodici SoftRT e NonRT -> minimizzare il tempo medio di risposta
  - Per i task aperiodici HardRT (task sporadici) -> *garantirne* il completamento entro la deadline previa accettazione

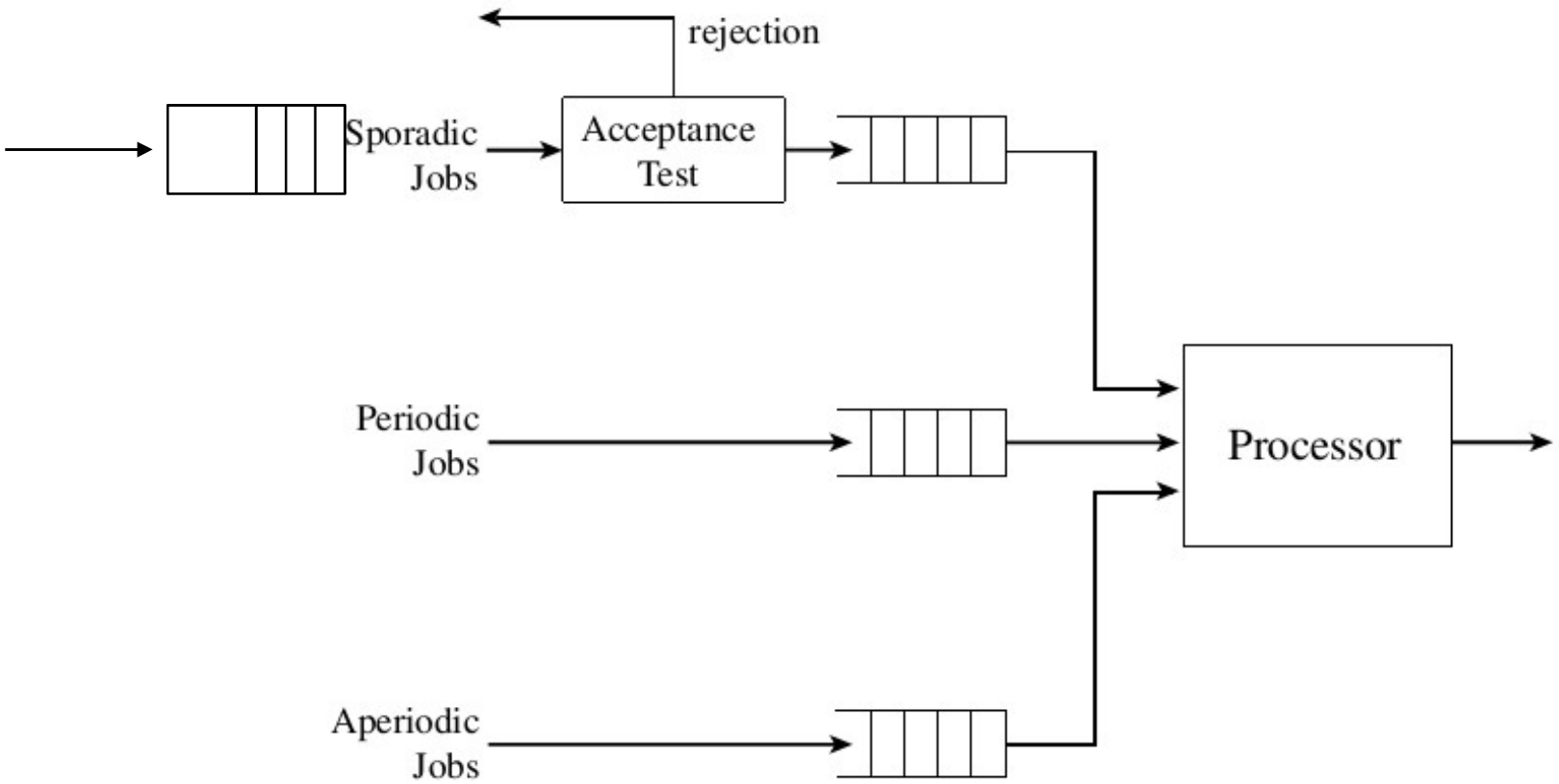
Introduzione

- Ipotesi generali: rilasciamo la sola ipotesi di periodicità di tutti i task, mentre manteniamo le altre →
  - Un solo processore
  - Preemption possibile ovunque, sia per task periodici che per task sporadici e aperiodici
  - Task periodici e job non periodici tutti indipendenti tra loro

Garanzia per carico non periodico

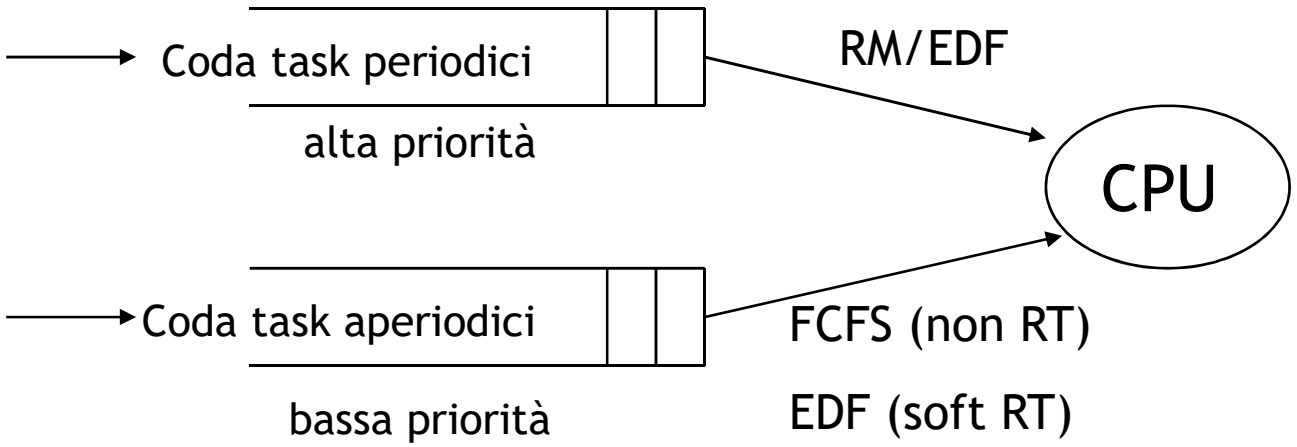
- Sul singolo job: → on-line
- Sull'intero task *non periodico* con istanze ripetute:
  - Per essere data in sede di prima attivazione occorre conoscere il tempo minimo tra due rilasci → task *sporadici*
  - Si assume talvolta che la deadline relativa sia il tempo minimo tra due rilasci

Architettura con code di priorità



Schedulazione in background

- I task aperiodici sono schedulati *quando il processore è libero* e non ci sono job periodici o sporadici pronti
- Problemi: possibile *starvation* in presenza di carico elevato dei task periodici, tempi di risposta elevati



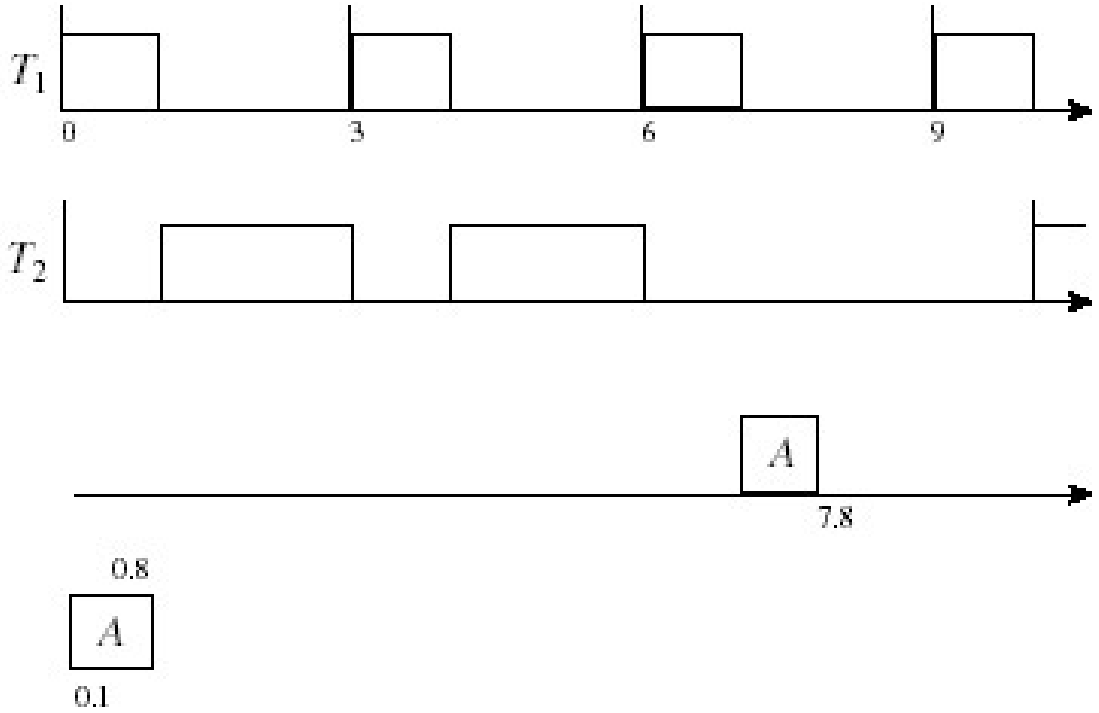
- Applicabile anche con altri algoritmi per i task periodici
- Strategie indipendenti per le due code

Tempo di risposta con schedulazione in background

T1=(3,1)

T2=(10,4)

A=aperiodico



- A rilasciato in t=0.1, con C_a=0.8 → f_a=7.8 e t_risp=7.7
- Prestazioni basse, potrebbe essere t_risp=0.8!

Schedulazione in background

- Idle time disponibile per l'esecuzione aperiodica in ogni iperperiodo H:

$$\Phi = (1- U)\ H$$

- Sia T un insieme di task periodici, con utilizzazione complessiva U e iperperiodo H. Un job aperiodico hard real-time Ja(Ca, Da), ove Da è la deadline relativa del job, è garantito se

$$\lceil Ca/\ \Phi \rceil\ H \leq Da$$

  - E' una condizione sufficiente
  - $\lceil Ca/\ \Phi \rceil$ è il numero di iperperiodi necessari per soddisfare la richiesta Ca
  - Utile solo per $Da \geq H$

Schedulazione in background

- Per m job aperiodici $J_1,\ldots, J_m$ hard real-time, ordinando l'insieme di task aperiodici per deadline assolute crescenti (→ EDF), l'insieme è garantito al tempo t se:

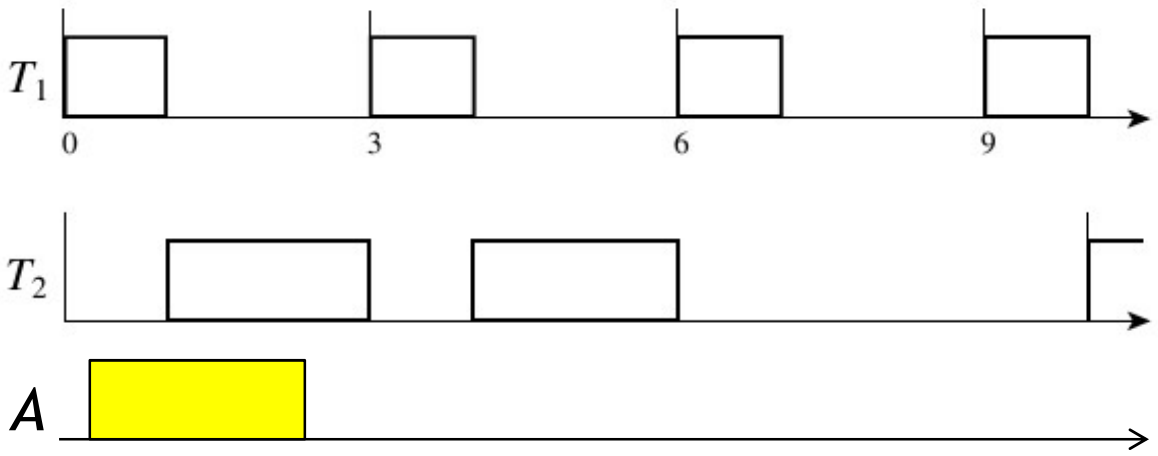
$$\forall h=1,\ldots,m \quad t + \lceil S_h / \Phi \rceil H \leq d_h$$

- ove: $S_h = \sum_{i=1,h} C_i(t)$, $C_i(t)$ = tempo residuo di esecuzione del job J_i all'istante t, d_h = deadline assoluta di J_h

- Risultato utile, come upper bound del tempo di risposta, anche per task soft real-time e non real-time

Schedulazione interrupt-driven

- Il job aperiodico viene posto in esecuzione non appena rilasciato, interrompendo l'esecuzione dei task periodici
- Minimizza il tempo di risposta dei job aperiodici
- Problema: possibili deadline miss per task periodici!
- Es.: T1=(3,1), T2=(10,4), A=(R_A=0.1, E_A=2.1)
- Deadline miss:
  - J11
  - J21



Schedulazione interrupt-driven

- In generale *non appropriata nei sistemi real-time*, se non per garantire uno o pochi task aperiodici/sporadici rilasciati mediante interrupt in un contesto di task periodici soft o non real-time
- Talvolta utilizzata in sistemi con carico real-time contenuto se il task aperiodico/sporadico ha una durata molto breve (-> *overhead*)
- E' possibile aggiungere lo *slack stealing*, ma nei sistemi priority-driven la sua realizzazione è complicata
- Esempi precedenti (sched. interrupt-driven + slack stealing):

  se E_A=0.8 -> T_{risp}=0.8

  se E_A=2.1 -> T_{risp}=9.0

Utilizzo di server per richieste aperiodiche

- Il tempo di risposta medio dei task aperiodici può essere migliorato, rispetto al servizio in background, *riservando un task periodico* al servizio delle richieste aperiodiche
- Denominato *Aperiodic server* o semplicemente *server;* oppure *Periodic server* (delle richieste aperiodiche)
- T_s = periodo del server, C_s = capacità del server (o budget)
- Il task *server* è schedulato con lo stesso algoritmo degli altri task periodici
- Il server $\tau_s(T_s,C_s)$ effettua il servizio delle richieste aperiodiche pendenti nei limiti della sua capacità C_s

Utilizzo di server per richieste aperiodiche

- L'ordine di servizio delle richieste aperiodiche pendenti è indipendente dall'algoritmo di scheduling dei task periodici
  - Ad es.: tempo di arrivo (FCFS), t. esec. (SJF), deadline (EDF)

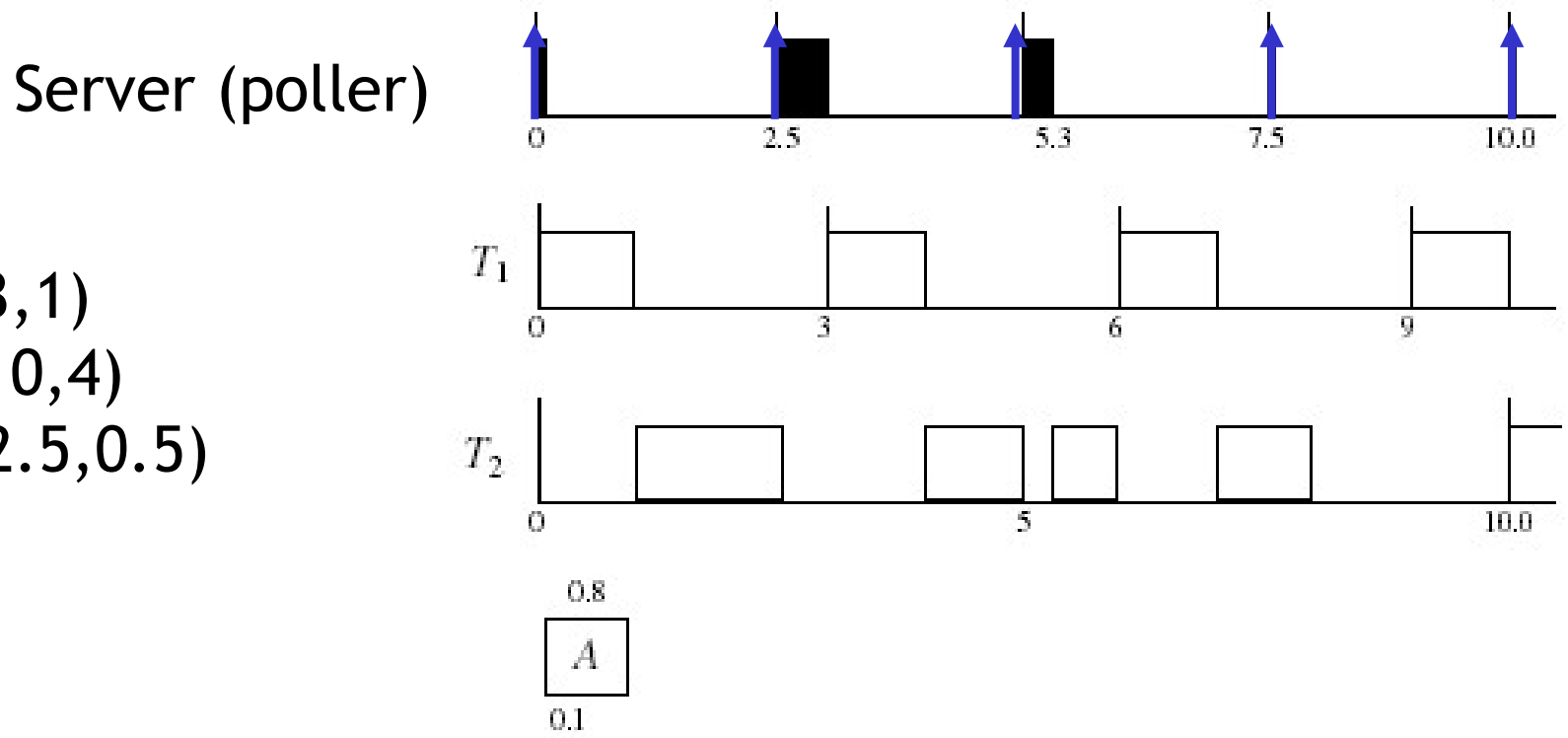
- L'esecuzione dei job aperiodici non deve pregiudicare la schedulabilità dei task periodici. I task periodici tuttavia *possono essere ritardati*, a differenza del background

- Due classi: server adatti sia per schedulatori a priorità statica che a priorità dinamica (ad es. Polling Server, Deferrable Server) e server abbinati a schedulatori a priorità dinamica (ad es. Dynamic Priority Exchange)

Polling server

- Il server $\tau_s(T_s,C_s)$ esegue le richieste aperiodiche pendenti nel suo istante di rilascio, e quelle che arrivano prima della sua conclusione, fino alla capacità C_s
- In assenza di richieste il server si sospende
- La capacità *non è preservata*, cioè non è disponibile per task aperiodici rilasciati successivamente entro il periodo del server
- Un job aperiodico *che arriva dopo che il server ha esaurito la coda del periodo corrente* sarà servito, se la capacità è sufficiente, nel periodo successivo
- La schedulazione dei task periodici, incluso il server, può essere sia RM che EDF

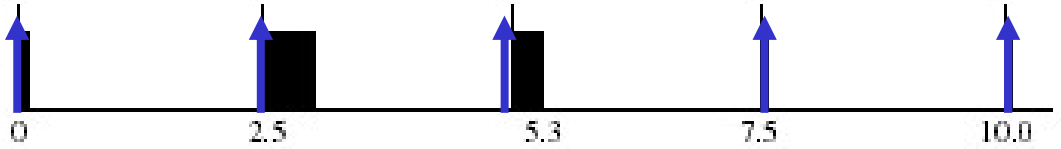
Esempio con polling server



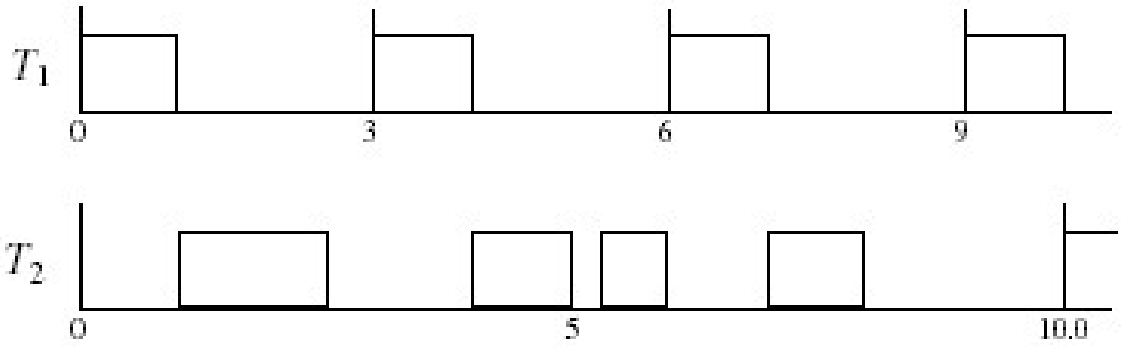
- Job aperiodico: A=(R_A=0.1,C_A=0.8)
- Con T_s=(2.5, 0.5), T_{risp}=5.2

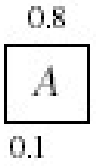
Esempio con polling server

Server (poller)



T1=(3,1)
T2=(10,4)
Ts =(2.5,0.5)





- Job aperiodico: A=(R_A=0.1,C_A=0.8)
- Con T_s=(2.5, 0.5), T_{risp}=5.2

Q: i task T_s, T_1, T_2 sono tutti garantiti?

Polling Server

- Polling Server si comporta nel peggiore dei casi come un task periodico (T_s, C_s)
- Test di garanzia:

$$\sum_{i=1,n} C_i/T_i + C_s/T_s \leq U_{lub}$$

- U_{lub} dipende dall'algoritmo di scheduling considerato e dai parametri del polling server
- Ad esempio, con RM e utilizzando il bound di Liu e Layland:

$$\sum_{i=1,n} C_i/T_i + C_s/T_s \leq U_{LL}(n+1)$$

Polling Server in presenza di task sporadici

- In presenza di task sporadici da garantire per tutte le istanze future, per essi si deve fare riferimento alla deadline relativa
- Nel caso più generale sono presenti task periodici, sporadici ed aperiodici
- Test di garanzia con task sporadici (gestiti individualmente come i periodici) e task soft RT gestiti con Polling Server:

$$\sum_{i\ period} Ci/Ti + \sum_{j\ sporad} Cj/Dj + Cs/Ts \leq U_{lub}$$

- Eventuali task aperiodici di tipo hard devono essere garantiti valutando la *capacità totale di server* disponibile fino alle deadline specificate

Polling server per garantire un task sporadico

- Esercizio:
- Proporre le condizioni sufficienti per garantire *un singolo task sporadico* mediante *un polling server* con parametri (T_s, C_s)
- Sia $\tau_A(T_A,D_A,C_A)$ il task sporadico, in cui T_A è il tempo minimo di interarrivo, D_A la deadline relativa, C_A il tempo di esecuzione di caso peggiore

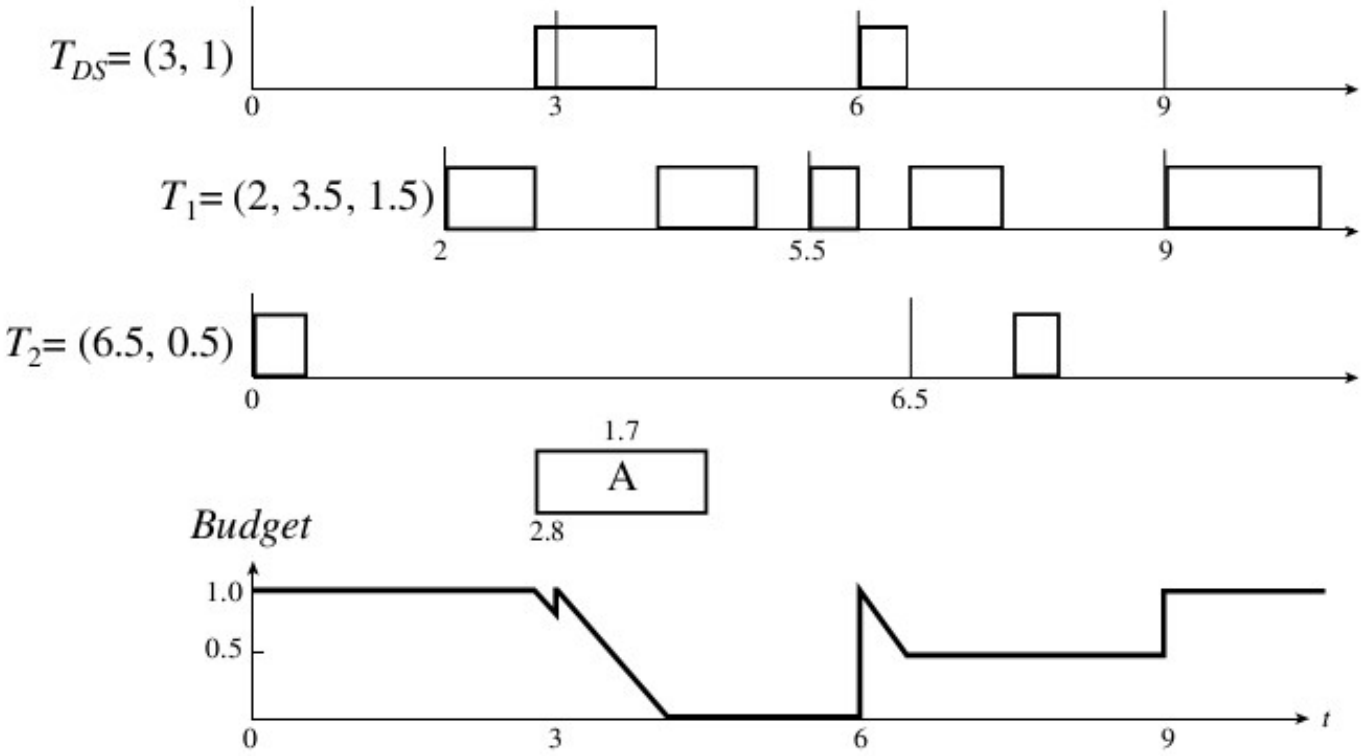
- «Sappiamo che τ_A può arrivare e vogliamo garantirci la possibilità di gestirlo rispettando la deadline»

Deferrable Server

- Deferrable Server (DS): Server periodico, in genere ad alta priorità, dedicato al servizio di attività aperiodiche; utilizzato spesso con RM, utilizzabile anche con EDF
- DS *conserva* la sua capacità anche se, quando è rilasciato, non ci sono richieste pendenti → DS può fornire un servizio immediato alle richieste
- DS conserva tutta la capacità per tutta la durata del periodo T_s: le richieste possono essere servite in ogni istante con la priorità del server, purché la capacità non sia esaurita
- All'inizio di ogni periodo la capacità è ripristinata al valore nominale; quella inutilizzata del periodo precedente è persa

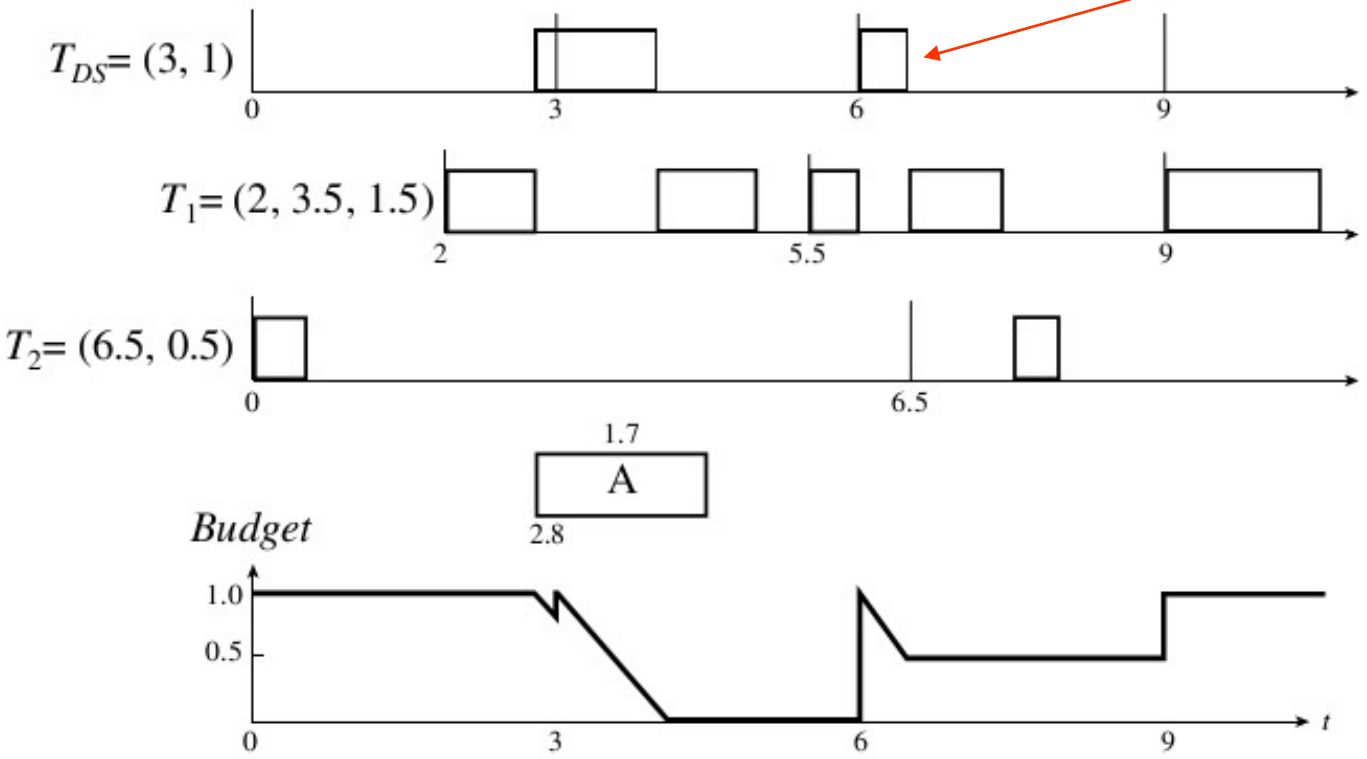
Deferrable Server

- Esempio: $\tau_{DS}(3,1)$, $\tau_1=(\phi_1=2,3.5,1.5)$, $\tau_2=(6.5,0.5)$
- Schedule RM:



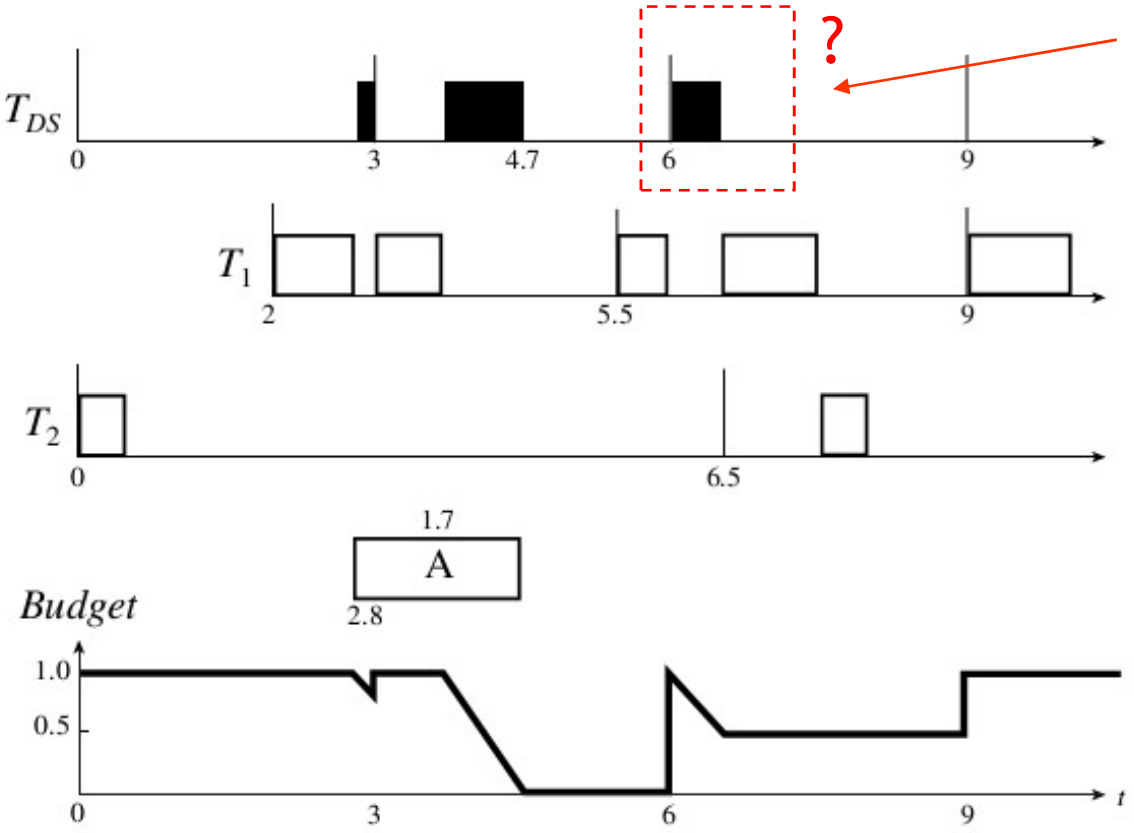
Deferrable Server

- Esempio: $\tau_{DS}(3,1)$, $\tau_1=(\phi_1=2,3.5,1.5)$, $\tau_2=(6.5,0.5)$
- Schedule RM:



Deferrable Server

- Esempio: $\tau_{DS}(3,1)$, $\tau_1=(\phi_1=2,3.5,1.5)$, $\tau_2=(6.5,0.5)$
- Schedule EDF:



T_{DS} e T_1 stessa deadline t=9

Deferrable Server

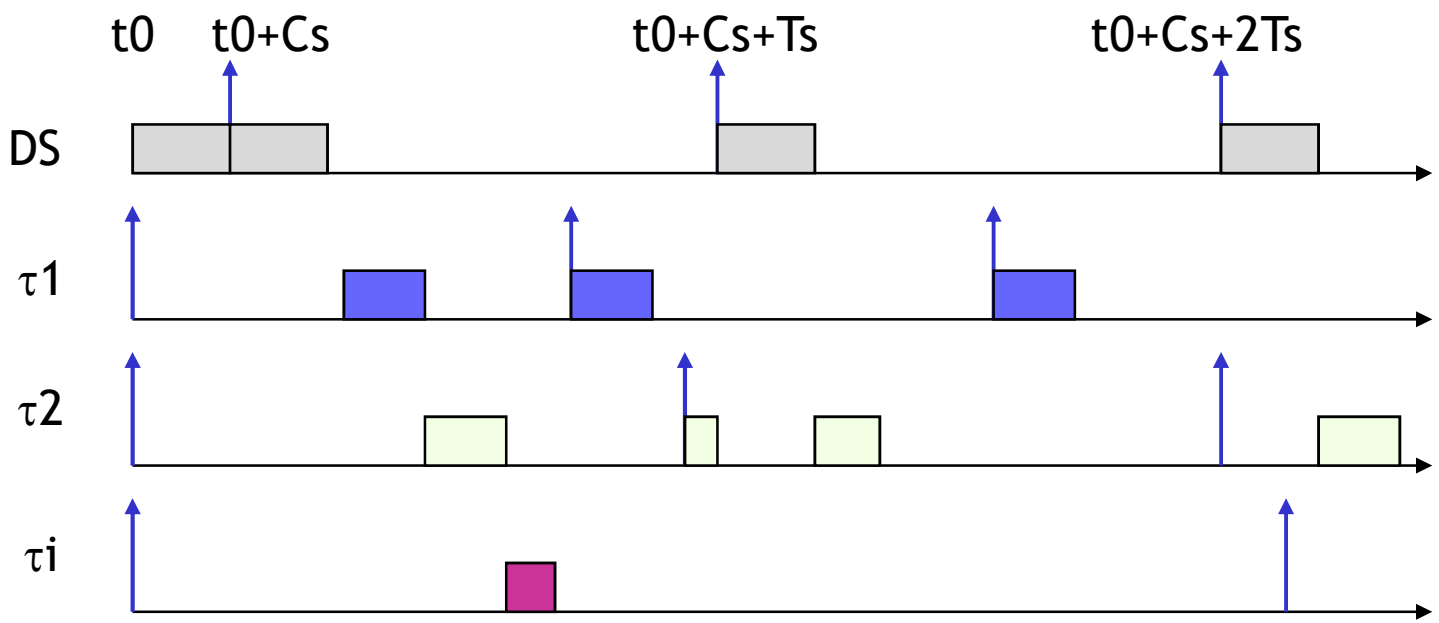
- Il tempo medio di risposta con DS è in genere nettamente migliore rispetto a PS
- DS tuttavia *non rispetta le ipotesi dei task periodici RM* (e in generale *work conserving*), in quanto il server è un task che non è sempre pronto all'inizio di ogni suo periodo
- La sua esecuzione può essere posticipata (deferred) o sospesa a causa della mancanza di richieste aperiodiche
- Il contributo di τ_s alla utilizzazione necessaria per garantire i task periodici è quindi superiore a C_s/T_s !

Deferrable Server

- In un sistema *a priorità fissa* in cui Di $\leq$ Ti $\forall$i e in cui è presente un server DS (T_s,C_s) *con priorità massima* tra tutti i task, l'istante critico per un task periodico τ_i si verifica in un istante t_0 in cui:
  - è rilasciato un job $J_{i,c}$ di τ_i
  - in t_0 sono rilasciati job di tutti i task a priorità maggiore
  - il server ha ancora capacità C_s in t_0, e in t_0 sono rilasciati job aperiodici tali da saturarlo
  - il successivo istante di ripristino della capacità del server è all'istante t_0+C_s

- NB: Vale per tutti gli assegnamenti statici di priorità

Deferrable Server



- Intervallo critico in presenza di DS: temporizzazione che determina il massimo tempo di risposta per τ_i

Deferrable Server

- L'analisi dell'intervallo critico evidenzia come *DS determini un carico maggiore* rispetto ad un task periodico di pari periodo e con tempo di esecuzione uguale alla capacità del server

- Per n task periodici schedulati in modo RM assieme ad un DS(Ts,Cs) con *periodo Ts arbitrario* e priorità RM, è possibile dare conto del *tempo di blocco* aggiuntivo che subisce *un task i a priorità inferiore* a quella del server considerando per esso Ui=(Ci+Bi)/Ti=(Ci+Cs)/Ti
- → condizione sufficiente per schedulabilità di τ_i:

$$\sum_{j=1,i} Cj/Tj + Cs/Ts + Cs/Ti \leq U_{RM}\,(i+1)$$

Deferrable Server

- Condizione sufficiente per schedulabilità di un task generico τ_i in presenza di un deferrable server DS e scheduling RM (DS con priorità RM) con priorità superiore a τ_i:

$$\sum_{j=1,i} Cj/Tj + Cs/Ts + Cs/Ti \leq U_{RM}(i+1)$$

- Osserviamo che:
  - il blocco dovuto al DS si aggiunge ad un eventuale blocco per sezioni non revocabili a priorità inferiore
  - i task con priorità superiore a DS non sono influenzati
  - sono presenti n+1 task
  - occorre verificare se anche DS è garantito
  - il test va eseguito task per task, non è un bound in forma chiusa
  - il bound RM fornisce una condizione in generale solo sufficiente

Deferrable Server

- Esiste un bound in forma chiusa per il caso RM integrato da un DS solo sotto specifiche ipotesi sui valori dei periodi dei task periodici in relazione a quello del server e se il DS è il task a rate massimo e massima priorità
- In generale, conviene aggiungere al DS un BS (background server) che possa proseguire l'elaborazione dei job aperiodici, in presenza di slack time, anche dopo che DS ha esaurito la capacità
- Mediante l'analisi time-demand, in un sistema a priorità statica il DS può essere utilizzato e garantito anche sotto ipotesi più ampie, in particolare nel caso in cui il server abbia priorità arbitraria

Deferrable Server con EDF

- Dati n task periodici indipendenti e revocabili, in esecuzione con scheduling EDF assieme ad un DS(Ts,Cs) con periodo Ts arbitrario, ciascun task periodico τ_i rispetta le proprie deadline se:

$$\sum_{j=1,n} Cj/\min(Dj,Tj) + Cs/Ts + Cs/Ts \cdot ((Ts - Cs)/Di) \leq 1$$

- Nel caso Dj=Tj, la condizione di garanzia si può riscrivere come:

$$\sum_{j=1,n} Cj/Tj + Cs/Ts + Cs/Ti \cdot (1 - Us) \leq 1$$

- In pratica, nel caso EDF il tempo di blocco causato da DS subito *da ciascun task* non è Bi=Cs ma è Bi=Cs (1-Us)

Deferrable Server con EDF

- Con EDF il tempo di blocco causato da DS e subito *da ciascun task* non è Bi=Cs ma è Bi=Cs (1-Us), cioè è un po' inferiore rispetto ad RM
- Tuttavia il blocco può essere subito da *tutti i task*, mentre nel caso RM solo i task a priorità inferiore di DS sono influenzati
- Poiché il DS preserva la banda fino alla fine del suo periodo, le ipotesi del teorema di Baker sono violate, ed il DS può ritardare qualsiasi task
- In presenza di più deferrable server con EDF è possibile considerare ciascuno di essi in modo additivo, ripetendo gli ultimi due termini della formula

Sporadic server

- Nella versione più semplice, lo Sporadic Server viene utilizzato assieme ad RM
- La capacità viene ripristinata non all'inizio di ogni periodo ma solo dopo che è stata consumata, parzialmente o totalmente, da una richiesta aperiodica
- Il server dal punto di vista computazionale si comporta esattamente come un task di pari periodo e capacità
- Vale pertanto la seguente formula di garanzia:

$$\sum_{i=1,n} C_i/T_i + C_s/T_s \leq U_{lub}$$

Altri server aperiodici

- Esistono molti altri tipi di server, alcuni dei quali abbinabili solo a schedulazione di tipo deadline-driven
  - *Server a priorità dinamica*

- Il *tradeoff* è tra la capacità del server di garantire un maggior numero di task periodici e aperiodici e la complessità realizzativa, ovvero l'overhead del server

Ripasso: test di garanzia per PS e DS

- Polling Server con RM:

$$\sum_{i=1,n} C_i/T_i + C_s/T_s \leq U_{RM}(n+1)$$

- Polling Server con EDF:

$$\sum_{i=1,n} C_i/T_i + C_s/T_s \leq 1$$

- Deferrable Server con RM:

$$\sum_{j=1,i} Cj/Tj + Cs/Ts + Cs/Ti \leq U_{RM}\,(i+1)$$

per ogni τ_i con priorità inferiore a τ_{DS}

- Deferrable Server con EDF:

$$\sum_{j=1,n} Cj/Tj + Cs/Ts + Cs/Ti \cdot (1 - Us) \leq 1 \quad \text{per ogni } \tau_i$$